Robotics
Lab 4: Object Recognition

Group 3
Sebastian Grasdijk (s1972243)

January 31, 2016

Exercise 1: Learning to use Theano(preparation)

Multiple hidden layers

Parameter	Value
Learning rate(η)	0.01
L1 regularization	0.00
L2 regularization	0.0001
Batch Size	20
Max epochs	1000
Hidden nodes	500

Table 1: The parameters used for the Neural Network

A neural network had to be altered in order to see what the results were when multiple layers were used. The MNIST handwritten digits dataset was used as the data set on which the neural network trained and tested. The parameters used can be seen in Table 1. Each of the layers added used 500 nodes. The output layer was a logistic regression layer with negative cost likelihood.

# of hidden layers	Validation error(in %)	Test error(in %)	Runtime(in min)
1	1.69	1.65	35.04
2	1.88	1.95	35.18
3	1.93	2.02	1564.34*

Table 2: The results obtained using different amounts of hidden layers. *This run was done on a laptop, which was shut down occassionally.

The results can be seen in Table 2. The learning algorithm potentially can run for 1000 epochs, but stops once it finished learning, which resulted in it never reaching 1000 epochs. Table 2 shows that the validation and test error increase when using additional hidden layers. This might be due to overfitting, because the added layers have the same amount of input and outputs of the layer before them, leading to creating a complex system for the problem at hand. This believe is further evidenced by the error increase for the amount of hidden layers used.

Linear Output layers with Mean Squared Error

The logistic regression output layer with negative cost likelihood is replaced by a linear output layer with a mean squared error cost function. It runs for about 20 epochs, where the mean

squared error goes from 120 to 76.

Exercise 2: Training a Multi-Layer Convolutional Neural Net

A convolutional neural network is used as a basis. The activation functions are replaced by Rectified Linear Units. Furthermore is local cost normalization added after the convolution layer and before the max pooling function.

Parameter	**Value**
Learning rate(η)	0.1
# of kernels	[20 50]
L2 regularization	0.0001
Batch Size	500
Max epochs	200

Table 3: The parameters used for the neural network

The same values as for the base convolutional neural network. These can be seen in Table 3. The neural network has two convolutional layers with a hidden layers. The hidden layer has 800 inputs with 500 outputs. Finally is there a logistic regression layer which gives 10 outputs, 1 for every digit. The same MNIST handwritten digits dataset was used as for assignment 1.

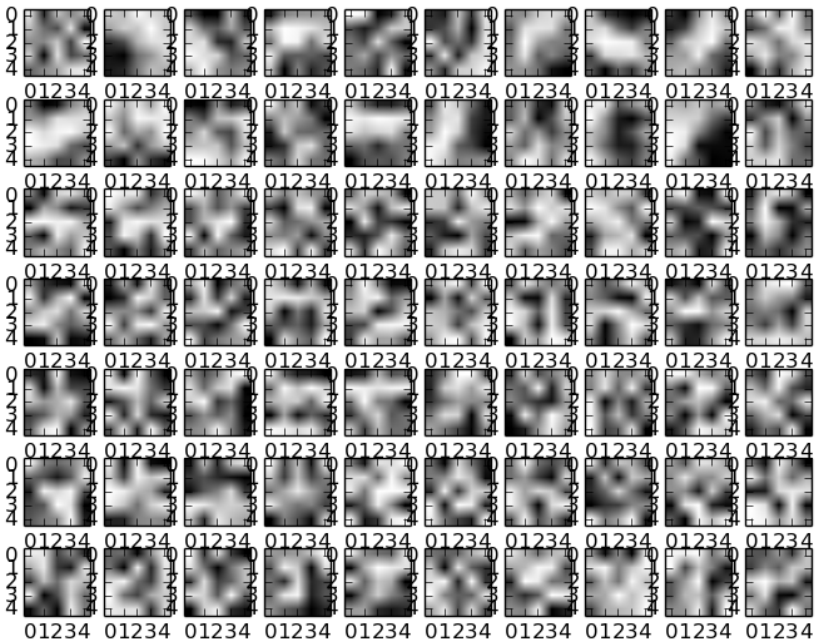


Figure 1: The final learned kernels for the convolutional neural network.

The results can be seen in Table 4. The neural network took 52.03 minutes to learn. Since there are 10 classes, the chance for a random correct classification is 10%, which would mean the test

Epoch	Validation error	Test error
1	18.14%	18.68%
49	1.58%	1.58%

Table 4: The results for the convolutional neural network.

error would be around 90%. As can be seen in Table 4, the resulting test error was 1.58, which is a lot below what the random test error is.

The Figure 1 shows the 70 kernels. Of which the first 20 kernels show the first layer of the convolutional layer, and the second shows the first 50 for the second layer. The first layer shows clear local contrast, due to the clear borders between the dark and light colours in the graphs.

Exercise 3: Training the CNN on a real dataset

The final assignment used the code from the second assignment as basis, but a different dataset. The CIFAR-10 dataset was used, a dataset with 10 distinct classes such as airplane, dog or frog.

Input

A different dataset meant that the format of the images also was different. The previous MNIST dataset used gray scale images which are 28 by 28 digits, while the CIFAR-10 dataset used colour images which are 32 by 32 pixels.

Therefore the data first had to be reshaped in order to be able to be parsed by the neural network. The size of the filter also had to be changed, as well as the kernel sizes had to be increased from 4x4 to 5x5.

The dataset was split into six batches, five training batches and one test batch. For ease of use is the fifth training batch used as validation set, giving us a balanced training/validation/test set.

Architecture

A number of settings for the convolutional neural network were tested in order to see which architecture gave the best scores. The size of nodes per hidden layers, the amount of hidden layers and amount of kernels were varied in order to see which gave the best result.

One of the issues we came across was a problem of overfitting. One of the methods to reduce overfitting available is to use dropout [1][2] . For each hidden layer used, a random mask of ones and zeroes is made and multiplied with the output of the layer. This randomly disables nodes while training in a hidden layer, essentially thinning the network temporarily by preventing those nodes into having incoming or outgoing connections.

Increasing the amount of kernels generally did not improve the results significantly, while increasing the runtime. Increasing the amount of nodes decreased the error percentages, up to a point. A too large network quickly became too complex and resulted in bad scores and long runtimes. Implementing the dropout method allowed the convolutional neural network to go from a test error of around 50% to 33%, which is quite an increase.

The results obtained and that are shown are using two almost equal architectures. The main difference between the two is that one uses local contrast normalization, while the other does not.

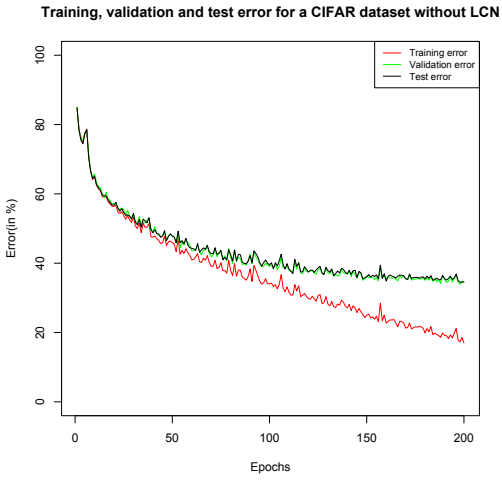
The architecture is as follows: first the image goes into the first convolution layer with 32 kernels, where the image is filtered and reduced from 32 by 32 to 14 by 14. The second convolution layer has 64 kernels and reduces the image size even further to a shape of 5 by 5.

This is flattened and then sent to the first hidden layer with rectified linear units. This has 1600 inputs and reduces the data to 1250 outputs. The second hidden layer also uses rectified linear units and reduces the data further, from 1250 inputs to 128 outputs.

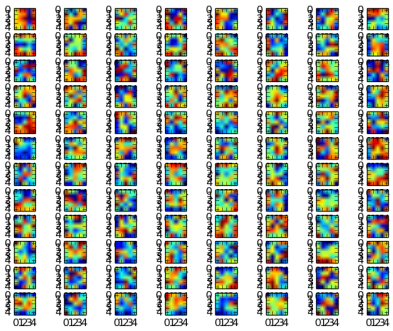
Finally the output layer consists of logistic regression, which has ten potential outputs for each of the classes.

Results

The results are presented in a graph, showing the change in the training, validation and test error over the epochs. Furthermore is there a graph with all of the kernels visualized. The results from two full runs are shown, one with local contrast normalization and one without. This allows us to also compare the effects of having local contrast normalization.



(a) The training, validation and test errors for a run over the number of epochs ran.



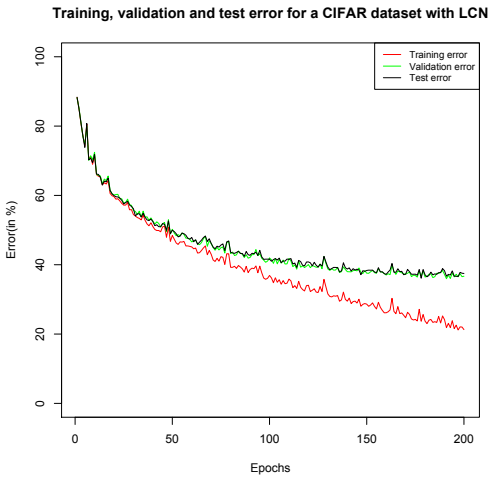
(b) The resulting kernels for a run.

Figure 2: The results for a run without local contrast normalization.

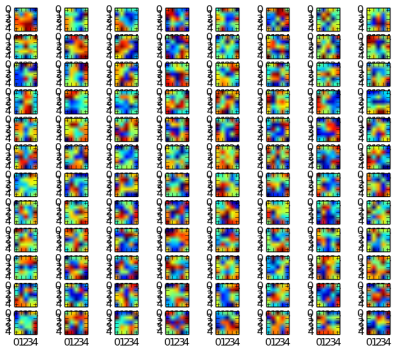
The figures 2b and 3b show the kernels when using and abstaining from using local contrast normalization. As can be seen, there doesn't appear to be a large difference in the results.

This insignificant difference for using local contrast normalization also appears when looking at the difference in errors. Not using local contract normalization even gave us a better training error with about the same validation and test errors.

The largest difference however, was the training time the algorithm needed to learn. Not using local contrast normalization took the system 34.13 minutes to learn and run 200 epochs, while if we were using local contrast normalization, the system took 201.16 minutes. Occam's razer dictates that the simpler model for the same results is better, and if you factor in learning time,



(a) The training, validation and test errors for a run over the number of epochs ran.



(b) The resulting kernels for a run.

Figure 3: The results for a run with local contrast normalization.

then the best approach would be to not use local contrast normalization, even though it should have a positive effect on our results.

So this unfortunately begs the question whether our implementation of the learning curve optimalization was correct. Exercise 2 showed kernels with were in line with the expectated kernels when using learning curve optimalization, and those were reused for the third assignment, yet the kernels appear to not really differ between the runs with and without local contrast optimalization.

Nevertheless are the results good. A good architecture for the convolutional neural network on the CIFAR-10 dataset was able to get an 18% test error according to the site where the CIFAR-10 dataset is located. With a fairly simple architecture and a non-optimized network due to time constraints, as well as missing performance boosting methods such as momentum, were we able to get a result of 32%.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[2] Nitish Srivastava et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.